Applying oracles of on-demand accuracy in two-stage stochastic programming – a computational study

Christian Wolf* Csaba I. Fábián† Achim Koberstein‡ Leena Suhl§

Abstract

Traditionally, two variants of the L-shaped method based on Benders' decomposition principle are used to solve two-stage stochastic programming problems: the single-cut and the multi-cut version. The concept of an oracle with on-demand accuracy was originally proposed in the context of bundle methods for unconstrained convex optimzation to provide approximate function data and subgradients. In this paper, we show how a special form of this concept can be used to devise a variant of the L-shaped method that integrates the advantages of the traditional variants while avoiding their disadvantages. On a set of 104 test problems, we compare and analyze parallel implementations of regularized and unregularized versions of the algorithms. The results indicate that significant speed-ups in computation time can be achieved by applying the concept of on-demand accuracy.

Keywords. Stochastic programming, two-stage problems, decomposition, bundle methods.

1 Introduction

Decomposition is an effective and time-honoured means of handling two-stage stochastic programming problems. It can be interpreted as a cutting-plane scheme applied to the first-stage variables. Traditionally, there are two approaches: one can use a disaggregate or an aggregate model. A major drawback of the aggregate model is that an aggregate master problem can not contain all the information obtained by the solution of the second-stage problems. The disaggregate master problem, on the other hand, may grow excessively. It is not easy to find a balance between the effort spent in solving the master problem on the one hand, and the second-stage problems on the other hand. The computational results of [20] give insights into this question.

In this study we report our experiments with a special inexact convex programming method applied to the aggregate master problem of the two-stage stochastic programming decomposition scheme. The convex programming method is of the type that uses an oracle with on-demand accuracy, a concept proposed by Oliveira and Sagastizábal [18]. We are going to use a special form discussed in [8], which, when applied to two-stage stochastic programming problems, is shown to integrate the advantages of the aggregate and the disaggregate models. We also examine the on-demand accuracy idea in an un-regularized context, which results a pure cutting-plane method in contrast to the level bundle methods treated in [18].

The paper is organized as follows. In Section 1.1 we outline the on-demand accuracy approach to convex programming, and present an algorithmic sketch of the partly asymptotically exact level method. In Section 2 we overview two-stage stochastic programming models and methods. Specifically, in Section 2.1 we sketch a decomposition method for two-stage problems based on the partly asymptotically exact level method. Section 3 discusses implementation issues. Our computational results are reported in Section 4, and conclusions are drawn in Section 5.

*DS&OR Lab, University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany. E-mail: christian.wolf@dsor.de.

†Department of Informatics, Kecskemét College, 10 Izsáki út, 6000 Kecskemét, Hungary. E-mail: fabian.csaba@gamf.kefo.hu.

‡Goethe University Frankfurt, Grüneburgplatz 1, 60323 Frankfurt am Main, Germany. E-mail: koberstein@wiwi.uni-frankfurt.de. Corresponding author.

§DS&OR Lab, University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany. E-mail: suhl@dsor.de.

1.1 Convex programming: applying oracles of on-demand accuracy

Let us consider the problem

$$\begin{array}{rl} \min & \varphi(\boldsymbol{x}) \\ \text{such that} & \boldsymbol{x} \in X, \end{array}$$

where $\varphi : \mathbb{R}^n \to \mathbb{R}$ is a convex function, and $X \subset \mathbb{R}^n$ is a convex closed polyhedron. We assume that φ is Lipschitz continuous over X with the constant Λ.

Oliveira and Sagastizábal [18] developed special regularization methods for unconstrained convex optimization, namely, bundle level methods that use oracles with on-demand accuracy. The methods work with approximate function data, which is especially useful in solving stochastic problems. Approximate function values and subgradients are provided by an oracle with on-demand accuracy. The accuracy of the oracle is regulated by two parameters: the first is a descent target, and the second is a tolerance. If the estimated function value reaches the descent target, then the prescribed tolerance is observed. Otherwise the oracle just detects that the target can not be met, and returns rough estimations of the function data, disregarding the prescribed tolerance. The method is based on [15], [13], and [7]; and integrates the level-type and the proximal approach.

In this paper we are going to use a special, non-proximal, form of the on-demand accuracy level bundle method of Oliveira and Sagastizábal. This is a partly asymptotically exact level method that uses exact evaluations when the descent target is met. The method is discussed in [8]. It is a cutting-plane scheme, using the level regularization of Lemaréchal, Nemirovskii and Nesterov [15].

We assume that the feasible polyhedron X is bounded. At iteration i, the descent target $\overline{\phi}_i$ will be the best function value known at this stage. An iterate meeting the descent target will be called *substantial*. If $\boldsymbol{x}_j$ is a substantial iterate then the oracle returns a subgradient: $l_j(\boldsymbol{x}_j) = \varphi(\boldsymbol{x}_j)$. Otherwise the oracle just detects that the target can not be met, and returns rough estimations of the function data.

Algorithm 1 *A partly asymptotically exact level method using exact evaluations in substantial points.*

1.0 Initialization.

Set the stopping tolerance $\epsilon > 0$.

Set the level parameter λ $(0 < \lambda < 1)$.
Set the accuracy regulating parameter γ such that $0 < \gamma < (1-\lambda)^2$.

Let $\overline{\phi}_0 = +\infty$ (upper bound for optimum).

Let $i = 1$ (iteration counter).

Find a starting point $\boldsymbol{x}_1 \in X$.

1.1 Bundle update.

If $i = 1$ then let $\delta_i > 0$, arbitrary. Otherwise let $\delta_i = \gamma\Delta_{i-1}$.

Let $l_i(\boldsymbol{x})$ be a linear function such that

(i.) $l_i(\boldsymbol{x}) \le \varphi(\boldsymbol{x}) \quad (\boldsymbol{x} \in X)$,

(ii.) $\|\nabla l_i\| \le \Lambda$, and

(iii.) either $l_i(\boldsymbol{x}_i) = \varphi(\boldsymbol{x}_i)$ or $l_i(\boldsymbol{x}_i) + \delta_i \ge \overline{\phi}_{i-1}$.

1.2 Near-optimality check.

Compute $\overline{\phi}_i = \min\limits_{1 \le j \le i} \varphi(\boldsymbol{x}_j)$.

Let $\underline{\phi}_i = \min\limits_{\boldsymbol{x} \in X} \varphi_i(\boldsymbol{x})$, where $\varphi_i(\boldsymbol{x}) = \max\limits_{1 \le j \le i} l_j(\boldsymbol{x}_j)$ is the current model function.

Let $\Delta_i = \overline{\phi}_i - \underline{\phi}_i$. If $\Delta_i < \epsilon$ then near-optimal solution found, stop.

1.3 Finding a new iterate.

Let $\boldsymbol{x}_{i+1}$ be the projection of $\boldsymbol{x}_i$ onto $X_i = \left\{\boldsymbol{x} \in X \,|\, \varphi_i(\boldsymbol{x}) \leq \underline{\phi}_i + \lambda\Delta_i\right\}$.

Increment i, and repeat from step *1.1*.

In step *1.3,* above, the projection of $\boldsymbol{x}_i$ onto X_i means finding the point in X_i nearest to $\boldsymbol{x}_i$. It means solving a convex quadratic programming problem.

Some of the iterations in the above method will be labeled *critical,* according to the following definition. Starting from $\boldsymbol{x}_1$, let us consider a maximal sequence of iterations $\boldsymbol{x}_1 \to \boldsymbol{x}_2,\ \ldots,\ \boldsymbol{x}_{s-1} \to \boldsymbol{x}_s$ such that $\Delta_1 \geq \Delta_2 \geq \cdots\cdots \geq \Delta_s \geq (1-\lambda)\Delta_1$ holds. Maximality of the sequence means that $\Delta_{s+1} < (1-\lambda)\Delta_1$. Then $\boldsymbol{x}_s \to \boldsymbol{x}_{s+1}$ will be labeled critical. The above construction is repeated starting from $\boldsymbol{x}_s$. Thus the iterations are grouped into sequences, and the sequences are separated with critical iterations.

Remark 2 *Concerning the practical efficiency of the (exact) level method of [15], Nemirovski in [17] (Chapter 5.3.2) observes the following experimental fact. When solving a problem of dimension n with accuracy ϵ, the level method makes no more than $n\ln(V/\epsilon)$ iterations, where V is a problem-dependent constant.*

This observation was confirmed by the experiments reported in [9] and [21], where the level method was applied in decomposition schemes for the solution of two-stage stochastic programming problems.

2 Two-stage stochastic programming models and methods

In the present discussion we assume discrete finite distributions, and linear functions. Moreover, we consider only problems with a bounded feasible domain and relatively complete recourse.

The first-stage decision is represented by the vector $\boldsymbol{x}$. Assume there are S possible outcomes (*scenarios*) of the random event, the sth outcome occurring with probability p_s. Suppose the first-stage decision has been made with the result $\boldsymbol{x}$, and the sth scenario has realized. The second-stage decision $\boldsymbol{y}$ is computed by solving the following *second-stage problem* or *recourse problem* that we denote by $\mathcal{R}_s(\boldsymbol{x})$.

$$\begin{array}{rl} \min & \boldsymbol{q_s^T y} \\ \text{such that} & T_s\boldsymbol{x} + W_s\boldsymbol{y} = \boldsymbol{h_s}, \\ & \boldsymbol{y} \geq \boldsymbol{0}, \end{array} \tag{1}$$

where $\boldsymbol{q_s}$, $\boldsymbol{h_s}$ are given vectors and T_s, W_s are given matrices. Let K_s denote the set of those $\boldsymbol{x}$ vectors for which the recourse problem $\mathcal{R}_s(\boldsymbol{x})$ has a feasible solution. This is a convex polyhedron. For $\boldsymbol{x} \in K_s$, let $q_s(\boldsymbol{x})$ denote the optimal objective value of the recourse problem. We assume that $q_s(\boldsymbol{x}) > -\infty$. The polyhedral convex function $q_s : K_s \to \mathbb{R}$ is called the *recourse function.*

The customary formulation of the *first-stage problem* is

$$\begin{array}{rl} \min & \boldsymbol{c^T x} + \sum\limits_{s=1}^{S} p_s\, q_s(\boldsymbol{x}) \\ \text{such that} & \boldsymbol{x} \in X, \\ & \boldsymbol{x} \in K_s \ \ (s = 1, \ldots, S), \end{array} \tag{2}$$

where $X := \{\boldsymbol{x} \,|\, A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$ is a non-empty polyhedron describing the constraints, $\boldsymbol{c}$ and $\boldsymbol{b}$ are given vectors and A is a given matrix, with compatible sizes. The expectation part of the objective, $q(\boldsymbol{x}) = \sum_{s=1}^{S} p_s\, q_s(\boldsymbol{x})$, is called the *expected recourse function.* This is a polyhedral convex function with the domain $K := K_1 \cap \ldots \cap K_S$.

The two-stage stochastic programming problem (2) - (1) can be formulated as a single linear programming problem called the *deterministic equivalent problem.*

In this paper we assume that the feasible domain X is bounded, and that $X \subset K$, hence the constraints $\boldsymbol{x} \in K_s$ $(s = 1, \ldots, S)$ are redundant in (2). Let us denote the dual of $\mathcal{R}_s(\boldsymbol{x})$ by $\mathcal{D}_s(\boldsymbol{x})$:

$$\begin{array}{rl} \max & \boldsymbol{z^T}(\boldsymbol{h_s} - T_s\boldsymbol{x}) \\ \text{such that} & W_s^T\boldsymbol{z} \leq \boldsymbol{q_s}, \end{array} \tag{3}$$

where $\boldsymbol{z}$ is a real-valued vector. The feasible region is a convex polyhedron that we assumed nonempty. Given $\boldsymbol{x} \in X$, the objective value is finite according to the assumption $X \subset K_s$.

Given a finite subset $\widetilde{U}_s$ of the feasible domain of $\mathcal{D}_s(\boldsymbol{x})$, the function

$$\widetilde{q}_s(\boldsymbol{x}) := \max_{\boldsymbol{u_s} \in \widetilde{U}_s} \boldsymbol{u_s^T}(\boldsymbol{h_s} - T_s\boldsymbol{x}) \quad (\boldsymbol{x} \in X) \tag{4}$$

is a lower approximation of $q_s(\boldsymbol{x})$ over X. Having appropriate subsets $\widetilde{U}_s$ for $s = 1, \ldots, S$, the *disaggregate-form* cutting-plane approximation of the first-stage problem (2) is constructed as

$$\begin{array}{rl} \min & \boldsymbol{c^T x} + \sum_{s=1}^{S} p_s \vartheta_s \\ \text{such that} & \boldsymbol{x} \in X, \quad \vartheta_s \in \mathbb{R} \ \ (s = 1, \ldots, S), \\ & \boldsymbol{u_s^T}(\boldsymbol{h_s} - T_s\boldsymbol{x}) \le \vartheta_s \ \text{ holds for any } \boldsymbol{u_s} \in \widetilde{U}_s \quad (s = 1, \ldots, S). \end{array} \tag{5}$$

The expectation in the objective, $\widetilde{q}(\boldsymbol{x}) = \sum_{s=1}^{S} p_s\, \widetilde{q}_s(\boldsymbol{x})$, is called the disaggregate model function. This is a lower approximation of $q(\boldsymbol{x})$ based on the sets $\widetilde{U}_s$ $(s = 1, \ldots, S)$.

An *aggregate form* of the first-stage problem (2) is

$$\begin{array}{rl} \min & \boldsymbol{c^T x} + \vartheta \\ \text{such that} & \boldsymbol{x} \in X, \quad \vartheta \in \mathbb{R}, \\ & \sum_{s=1}^{S} p_s \boldsymbol{u_s^T}(\boldsymbol{h_s} - T_s\boldsymbol{x}) \le \vartheta \ \text{ holds for any } (\boldsymbol{u_1}, \ldots, \boldsymbol{u_S}) \in \widetilde{\mathcal{U}}. \end{array} \tag{6}$$

where $\widetilde{\mathcal{U}} \subset \widetilde{U}_1 \times \cdots \times \widetilde{U}_S$ is a certain subset of the Cartesian product. Namely, each element of $\widetilde{\mathcal{U}}$ belongs to a (potential) facet in the graph of the function $\widetilde{q}(\boldsymbol{x})$. There may be facets not represented in $\widetilde{\mathcal{U}}$. The expectation in the objective,

$$f(\boldsymbol{x}) = \max_{(\boldsymbol{u_1}, \ldots, \boldsymbol{u_S}) \in \widetilde{\mathcal{U}}} \sum_{s=1}^{S} p_s \boldsymbol{u_s^T}(\boldsymbol{h_s} - T_s\boldsymbol{x}) \tag{7}$$

is called the aggregate model function. This is a lower approximation of the disaggregate model function $\widetilde{q}(\boldsymbol{x})$.

Remark 3 *By selecting basic solutions of the respective dual recourse problems into the sets $\widetilde{U}_s$ $(s = 1, \ldots, S)$, we can ensure that the model functions $\widetilde{q}(\boldsymbol{x})$ and $f(\boldsymbol{x})$ are Lipschitz continuous with a constant depending only on the data of the two-stage stochastic programming problem.*

In what follows we assume that the sets $\widetilde{U}_s$ $(s = 1, \ldots, S)$ consist of basic solutions.

The difference between the aggregate and the disaggregate problem formulations may result in a substantial difference in the efficiency of the solution methods. By using disaggregate cuts, more detailed information is stored in the master problem. This is done at the expense of larger master problems. Based on the numerical results of [3] and [10], Birge and Louveaux [4] conclude that the multicut approach is in general more effective when the number of the scenarios is not significantly larger than the number of the constraints in the first-stage problem. Results of the computational study [21] confirm that the scale-up properties of solvers based on aggregate models are better than those of solvers based on disaggregate models, though the break-even thresholds are generally high. The results of the computational study [20] provide further insights into the effects of cut aggregation.

2.1 Applying an oracle with on-demand accuracy

In order to apply the on-demand accuracy approach to two-stage stochastic programming problems, Oliveira and Sagastizábal [18] propose inserting a new solver component between the aggregate master problem and

the second-stage problems. The role of the new component is to provide approximate values and gradients of the expected recourse function, based on the information represented in $\widetilde{U}_s$ $(s = 1, \ldots, S)$.

In this paper we work with Algorithm 1 adapted to the aggregate master problem (6). An additional parameter κ is needed. Let us set $0 < \kappa < 1 - \lambda$, where λ is the level parameter in Algorithm 1. Let $\overline{D}$ denote the best objective value known at the present stage of the solution process. Let $\boldsymbol{x}_i$ denote the current iterate. We assume that $i > 1$, and the iteration $(i-1) \to i$ was non-critical. If

$$\boldsymbol{c}^T \boldsymbol{x}_i + \widetilde{q}(\boldsymbol{x}_i) \geq \kappa \Big\{ \boldsymbol{c}^T \boldsymbol{x}_i + f(\boldsymbol{x}_i) \Big\} + (1 - \kappa)\overline{D} \tag{8}$$

holds, then the aggregate model function is updated by adding a linear support function of $\widetilde{q}(\boldsymbol{x})$ at $\boldsymbol{x}_i$. This can be constructed without solving second-stage problems. Such a cut is legitimately added in Algorithm 1, i.e., the cut satisfies the criteria *(i), (ii), (iii)* of step *1.1,* with accuracy parameter set to $\gamma = \kappa(1 - \lambda)$. (See [8] for a proof.) The above procedure results

Algorithm 4 *A decomposition method for two-stage problems based on Algorithm 1.*

4.0 Initialization.

Set the stopping tolerance $\epsilon > 0$.

Set the level parameter λ $(0 < \lambda < 1)$.
Set the descent target parameter κ such that $0 < \kappa < 1 - \lambda$.

Let $\widetilde{U}_s = \emptyset$ $(s = 1, \ldots, S)$, and let $\widetilde{\mathcal{U}} = \emptyset$.

Let $i = 1$ (iteration counter).

Find a starting point $\boldsymbol{x}_1 \in X$.

4.1 Bundle update.

If $i = 1$ then

Solve the dual recourse problems $\mathcal{D}_s(\boldsymbol{x}_1)$ $(s = 1, \ldots, S)$;
let $\boldsymbol{u}_s$ $(s = 1, \ldots, S)$ be the respective optimal basic solutions.

Add $\boldsymbol{u}_s$ to $\widetilde{U}_s$ $(s = 1, \ldots, S)$, updating $\widetilde{q}$; and add $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_S)$ to $\widetilde{\mathcal{U}}$, updating f.

Otherwise, if the iteration $(i-1) \to i$ is non-critical and (8) holds, then

Construct a support function of $\widetilde{q}(\boldsymbol{x})$ at $\boldsymbol{x}_i$,
in the form $\sum_{s=1}^{S} p_s \, \widehat{\boldsymbol{u}}_s^T (\boldsymbol{h}_s - T_s \boldsymbol{x})$, with $\widehat{\boldsymbol{u}}_s \in \widetilde{U}_s$.

Add $(\widehat{\boldsymbol{u}}_1, \ldots, \widehat{\boldsymbol{u}}_S)$ to the set $\widetilde{\mathcal{U}}$, updating f.

Otherwise

Solve the dual recourse problems $\mathcal{D}_s(\boldsymbol{x}_i)$ $(s = 1, \ldots, S)$;
let $\boldsymbol{u}_s$ $(s = 1, \ldots, S)$ be the respective optimal basic solutions.

Add $\boldsymbol{u}_s$ to $\widetilde{U}_s$ $(s = 1, \ldots, S)$ updating $\widetilde{q}$; and add $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_S)$ to $\widetilde{\mathcal{U}}$, updating f.

4.2 Near-optimality check.

Let $\overline{D} = \min\limits_{1 \leq j \leq i} \{\boldsymbol{c}^T \boldsymbol{x}_j + q(\boldsymbol{x}_j)\}$. Let $\underline{D} = \min\limits_{\boldsymbol{x} \in X} \{\boldsymbol{c}^T \boldsymbol{x} + f(\boldsymbol{x})\}$.

Let $\Delta = \overline{D} - \underline{D}$. If $\Delta < \epsilon$ then near-optimal solution found, stop.

4.3 Finding a new iterate.

Let $\boldsymbol{x}_{i+1}$ be the projection of $\boldsymbol{x}_i$ onto $\big\{\boldsymbol{x} \in X \,|\, \boldsymbol{c}^T \boldsymbol{x} + f(\boldsymbol{x}) \leq \underline{D} + \lambda\Delta\big\}$.

Increment i, and repeat from step *4.1.*

3 Implementation

The Level decomposition method as well as the on-demand accuracy oracle were implemented in an existing implementation of a parallel nested Benders (PNB) decomposition algorithm [20]. The PNB solver supports cut aggregation by specifying scenario partitions [19], but we only use the fully aggregated model. The parallel implementation allows to use all available cores, therefore solving subproblems as well as calling the on-demand accuracy oracle is done in parallel. For further implementation details regarding the basic Benders decomposition algorithm, please refer to [20]. In the following, we explain two particular aspects of algorithm (4), namely the implementation of the on-demand accuracy oracle in step 4.1 and the level projection problem in step 4.3.

In order to perform the test if equation (8) holds in step 4.1, the disaggregated model function $\tilde{q}(x_i) = \sum_{s=1}^{S} p_s \tilde{q}_s(x_i)$ must be computed. The stored basic solutions of $\mathcal{D}_s(\cdot)$, $u_s \in \tilde{U}_s$, are needed to compute $\tilde{q}_s(x_i) = \max_{\hat{u}_s \in \tilde{U}_s} \hat{u}_s(h_s - T_s x_i)$. After computing $u_s(h_s - T_s x_i)$ for every $u_s \in \tilde{U}_S$, the $\hat{u}_s \in \tilde{U}_s$ that maximizes $u_s(h_s - T_s x_i)$ can be found easily. The computation must be carried out in every iteration where equation (8) is checked, as the current iterate x_i changes from iteration to iteration.

If equation (8) holds and the current iteration is non-critical, the on-demand optimality cut $\sum_{s=1}^{S} p_s \hat{u}_s T_s x + \vartheta \geq \sum_{s=1}^{S} p_s \hat{u}_s h_s$ is added to the master problem (2). This is the same as adding $(\hat{u}_1, \ldots, \hat{u}_S)$ to $\tilde{\mathcal{U}}$.

As usual, the upper bound $\overline{D}$ is updated in each substantial iteration. The lower bound $\underline{D}$ is obtained by solving the aggregated master problem (6) in Step 4.2, whose solution we denote with x'. The next iterate x_{i+1} is defined by projecting the current master problem solution x' to a certain level set of the aggregated model function. This is done by solving the quadratic program

$$\begin{array}{rl} \min & ||\boldsymbol{x} - \boldsymbol{x'}||_2^2 \\ \text{such that} & \boldsymbol{x} \in X, \quad \vartheta \in \mathbb{R}, \\ & \sum\limits_{s=1}^{S} p_s \boldsymbol{u_s^T}(\boldsymbol{h_s} - T_s \boldsymbol{x}) \leq \vartheta \ \text{ holds for any } (\boldsymbol{u_1}, \ldots, \boldsymbol{u_S}) \in \widetilde{\mathcal{U}} \\ & \boldsymbol{c^T x} + \vartheta \leq \underline{D} + \lambda \Delta. \end{array} \tag{9}$$

The next iterate for the algorithm is the solution x, ϑ of the projection problem (9), which is denoted as x_{i+1}, ϑ_{i+1}.

Though the present study focusses on problems with complete recourse, our code can handle incomplete recourse also. No optimality cut is generated in case the current first-stage solution x_i is not feasible for the whole problem. Instead, we generate a feasibility cut of the usual form $u_s^T(h_s - T_s x_i) \leq 0$, where u_s denotes a ray of the feasible domain of $\mathcal{D}_s(x_i)$. (See, e.g., [4] for a detailed description.)

Thus if iterate i turns out to be substantial in course of step 4.1 of Algorithm 4, and any of the dual recourse problems $\mathcal{D}_s(x_i)$ $(s = 1, \ldots, S)$ proves unbounded, then we add a feasibility cut to the master problem. (The set $\widetilde{\mathcal{U}}$ is not updated in such cases.) – Of course this arrangement requires a final feasibility check at the conclusion of Algorithm 4.

This implementation differs from the one in [9], as the present regularization does not extend to feasibility issues.

4 Computational study

The computational results that were achieved by applying the on-demand accuracy approach in conjunction with level decomposition on a diverse set of two-stage stochastic programs with recourse are presented in this section.

4.1 Setup

All experiments were carried out on a Windows 7 machine with an 3.4 Ghz Intel i7-3770 processor with four physical cores, but eight logical cores due to hyper-threading and 16 GiB of RAM. The underlying LP

solver is the Cplex 12.4 dual simplex solver, with one thread. The Barrier method was used to solve the deterministic equivalents, with eight threads. The solution times are wall-clock times of the solution process, given in seconds, without the times for reading in the SMPS files. A time limit of 3,600 seconds was enforced for all solution runs. The expected value problem solution was chosen as the first stage initial solution. We set $\lambda = 0.5$ for all experiments and κ also. We evaluated the performance of level decomposition with on-demand accuracy (Level-ODA), level decomposition (Level), single-cut Benders (Benders-SC), multi-cut Benders (Benders-MC), single-cut Benders with on-demand accuracy (Benders-ODA) and the deterministic equivalent problem (DEQ).

The Benders-SC, Benders-MC and Benders-ODA algorithm are all unregularized methods, where the next iterate is determined by the solution x of the master problem. Level decomposition with $\lambda = 0$ results in the unregularized Benders decomposition method, as the solution set is restricted to the set of optimal solutions of the master problem. Thus algorithm 4 describes Benders-ODA, when λ is set to zero.

To compare the computing times, let $t_{p,m}$ be the solution time of method m on problem p and P be the number of problems. We give the arithmetic mean $1/P \sum t_{p,m}$, the geometric mean $(\prod t_{p,m})^{1/P}$ and the shifted geometric mean $(\prod t_{p,m} + s)^{1/P} - s$ of the whole test set, as a problem-wise comparison is not really helpful for so many instances. The arithmetic mean is sensitive to several instance that take a long time to solve. Performance differences among instances with lower computing times are then neglected. The geometric mean highlights differences in computing times, but if many instances take only a small amount of time, they dominate the results. The shifted geometric mean is a compromise between the arithmetic and the geometric mean, as the impact of small differences for small instances is reduced by the shifting parameter s, which we set to ten for all following results (see the discussion in [1]). We also use performance profiles [6] to provide a graphical representation of the performance of the algorithm on the whole test set.

4.2 Test Sets

We use a total of 105 different instances from different test sets, which we describe next. The *deak* test set [5] is available online at `http://web.uni-corvinus.hu/~ideak1/kut_en.htm`. The instances were also used in the proposal of the on-demand accuracy approach [18]. The *slptestset*, described in [2], is available at the homepage of A. Felt `http://www4.uwsp.edu/math/afelt/slptestset.html`. The *rand* problems are provided by [21]. Having been generated with the test problem generator of Kall and Mayer [12], they do not posses any real world structure, but can still be used to test for scale-up properties of algorithms. The sslp problems are contributed by L. Ntaimo and S. Sen. They are part of the stochastic integer test set library *SIPLIB*, which is compiled by S. Ahmed and is available online at `http://www2.isye.gatech.edu/~sahmed/siplib/`. For this computational study, we solved the LP relaxation. D. Holmes provides the *POSTS* testset online at `http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html`. It contains several two-stage and multi-stage problems. The numerically challenging *saphir* problems [14] and sampled versions of the instances contained in the testset used by Linderoth et al. [16] round up our testset. The dimensions of the test set are displayed in Table 11

4.3 Computational results

Feasibility issues caused no difficulty in course of the present experiments. Though our instances of the *saphir* problems are not of relatively complete recourse, all iterates became and remained feasible after $40-90$ initial feasibility cuts.

The performance profile of the on-demand accuracy approach is shown in Figure 1. It is a comparison level decomposition with on-demand accuracy, level decomposition, L-shaped method with single- and multi-cut, L-shaped method with on-demand accuracy, and the deterministic equivalent problem solved with Cplex. The profile shows that level decomposition with on-demand accuracy solves more than half of the problems faster than the other algorithms. In particular, it solves about 88 % of all problems within a factor of two of the fastest algorithms. The detailed computing times for every method and every problem are given in Table 12 at the end of the paper.

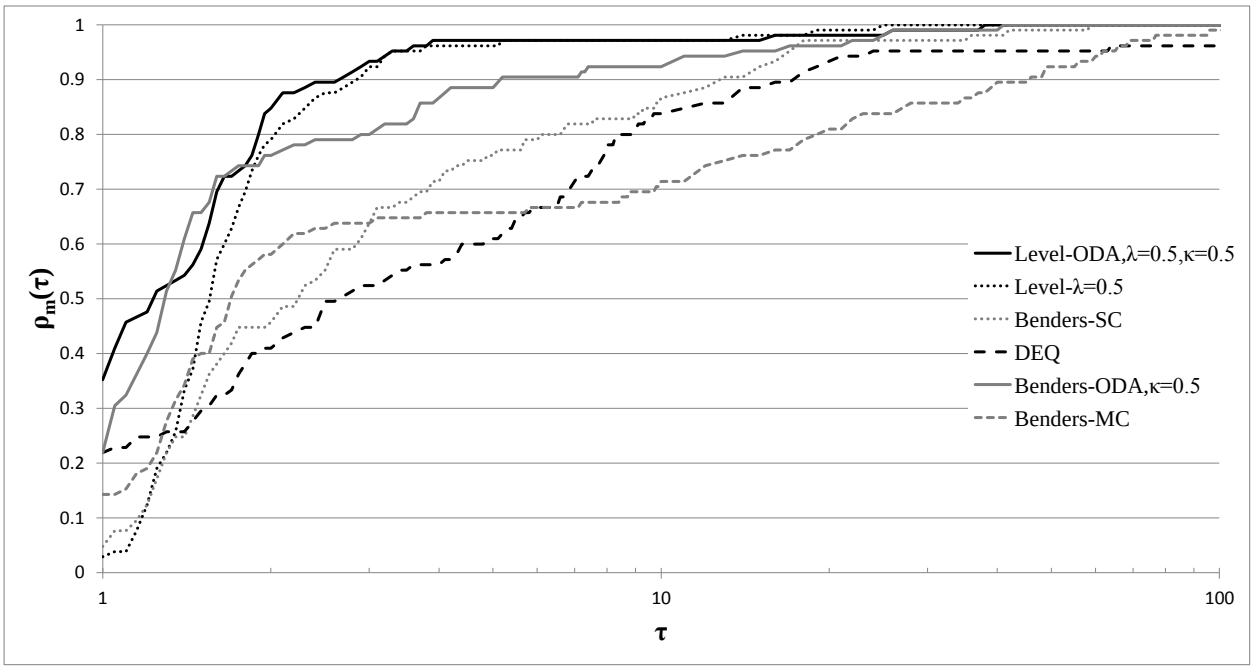

Figure 1: Performance profile of different algorithms on the test set.

The computational statistics for the different algorithms are given in Table 1. All three measures give the same efficiency order of the decomposition methods. Level decomposition with on-demand accuracy ranks first, followed by level decomposition, Benders decomposition with on-demand accuracy, single-cut Benders, and multi-cut Benders. The on-demand accuracy approach without regularization is notably slower than the regularized methods, but still faster than single-cut Benders alone, it takes 45 % of the computing time of single-cut Benders. Level decomposition with on-demand accuracy runs in only 21 % of the time needed by single-cut Benders. The geometric mean and the shifted geometric mean both show considerable improvements, 52 % and 40 %, respectively. This gap can be explained by two observations. First, it shows that level decomposition with on-demand accuracy is especially helpful in solving large instances, e.g., rand, LandS, 20_term, ssn, saphir, and 4node-32768. Second, the improvement of level decomposition with on-demand accuracy is not so large or negative on some smaller instances, e.g., 4node with up to 1024 scenarios, environ, and some instances from the deak test set.

Benders-MC ranks last according to all three measures. This is because the scale-up properties of the multi-cut method are worse than the scale-up properties of the single-cut methods. This fact was pointed out in [4], and confirmed by the test results of [21]. Our results in Table 12 also demonstrate the difference in the scale-up properties of single-cut and multi-cut methods. It must be noted, however, that much depends on the solver used for the solution of the master problem. [20] gives insights into this issue by presenting computational efforts spent in solving the first-stage and second-stage problems, respectively. In this paper we also present such data, in Table 4, below.

The comparison with the deterministic equivalent, which was solved with the parallelized barrier method of Cplex, confirms recent results that specialized solution methods are advantageous compared with solving the deterministic equivalent with standard solvers [21, 20].

The reason why on-demand accuracy speeds up the solution process can be seen by comparing the overall iteration and substantial iteration counts that were achieved by the different methods and are shown in Table 2 and 3, respectively. The on-demand accuracy approach has five percent more iterations, but just about 57% of substantial iterations, compared with the level method measured by the arithmetic mean. 45% of all iterations are insubstantial iterations, in which no subproblems have to be solved. These iterations are

Table 1: Computing times in seconds for different algorithms on the test set. The computing times are both given as absolute values and as as a percentage of the computing time of single-cut Benders decomposition (Benders-SC), for the arithmetic mean (AM), geometric mean (GM) and the shifted geometric mean (SGM). The compared methods are level decomposition with on-demand accuracy (Level-ODA), level decomposition (Level), multi-cut Benders decomposition (Benders-MC), Benders decomposition with on-demand accuracy (Benders-ODA) and the deterministic equivalent (DEQ). The Cplex barrier method solved four problems incorrectly, env-xlrge, saphir-100, saphir-1000, and ssn-3000. These problems were excluded for the computation of the results for the DEQ.

	AM	% Benders-SC	GM	% Benders-SC	SGM+10	% Benders-SC
Level-ODA	11.20	21	1.41	52	5.28	40
Level	14.92	28	1.66	61	6.42	48
Benders-SC	53.27	100	2.73	100	13.30	100
Benders-MC	263.60	495	3.62	132	21.33	160
Benders-ODA	24.08	45	1.69	62	8.74	66
DEQ	72.51	136	2.64	97	11.37	86

therefore quite fast, compared to a typical substantial iteration. The same holds for Benders decomposition with on-demand accuracy, where 84 % of all iterations are insubstantial iterations, but the increase in overall iterations is just 11 %, compared with Benders-SC. Although Benders-MC needs only about 7% of the iterations of Benders-SC, the computing time is much higher.

Table 2: Iteration counts for the whole test set.

	AM	% Benders-SC	GM	% Benders-SC	SGM+10	% Benders-SC
Level-ODA	78.83	29	51.99	62	55.58	59
Level	74.82	27	49.10	59	52.52	56
Benders-SC	274.38	100	83.26	100	94.52	100
Benders-MC	20.19	7	12.65	15	14.64	15
Benders-ODA	328.89	120	92.11	111	104.56	111

Table 3: Substantial iteration counts for the whole test set.

	AM	% Benders-SC	GM	% Benders-SC	SGM+10	% Benders-SC
Level-ODA	43.05	16	28.64	34	31.04	33
Level	74.82	27	49.10	59	52.52	56
Benders-SC	274.38	100	83.26	100	94.52	100
Benders-MC	20.19	7	12.65	15	14.64	15
Benders-ODA	53.25	19	25.37	30	29.23	31

Our results are in accordance with the practical efficiency estimate of the level method, cited in Remark 2. With the level parameter setting $\lambda = 0.5$, we found that there were generally less than n iterations between any two consecutive critical iterations. (n denotes the number of the first-stage variables.) This holds for both the Level and the Level-ODA methods. For each problem instance, we considered the maximal number of iterations occurring between any two consecutive critical iterations. This was then divided with the number of the first-stage variables. The ratios are very similar in case of the Level and the Level-ODA methods. The ratios fall below 1.0 in 93 problems from the 105 tested. In 6 further cases, the ratios fall

below 1.67. The corresponding problems belong to the saphir, gdb and 20_term problem schemes. The saphir problems need many iterations to reach a first-stage solution which is feasible for the whole problem. Without taking these iterations into account, the ratios would be also well below 1.0. In the remaining 6 cases, the ratios are below 8. The corresponding problems belong to the sslp problem schemes.

The iteration counts are reflected in the amount of time the algorithm spends at particular tasks, as can be seen in Table 4. The times are summed wall clock solution times over all threads. The starkest differences can be found between single-cut and multi-cut Benders. Where Benders-SC spent only a small amount of time solving the master problem, it spent much more time building and solving subproblems. On the other hand, Benders-MC spent most of the time solving the master problem, and the least amount of any algorithm on solving and building subproblems. This is in accordance with the iteration counts.

Comparing level decomposition with and without on-demand accuracy, the time spent in solving the master problems is a little bit higher, but this is to be expected due to the higher number of overall iterations. The same holds for the time spent in the projection problem. A substantial decrease can be found in the time spent in solving second stage subproblems as well as the time needed to build these subproblems, it takes 69% of level decomposition without on-demand accuracy. The time spent in generating on-demand accuracy cuts is relatively low, it amounts to 1% of the whole CPU time. The on-demand accuracy generation time is larger for Benders-ODA, as the number of iterations is much larger than for Level-ODA.

Table 4: Arithmetic mean of computing times spent at distinct tasks over the whole test set. The time for the tasks is wall clock time for each thread, i.e., summed over all threads.

Task	Stage	Level-ODA	Level	Benders-SC	Benders-MC	Benders-ODA
Solve Master	1	0.79	0.72	2.54	259.70	4.09
Solve Projection	1	0.58	0.54	0.00	0.00	0.00
Solve ODA	2	0.70	0.00	0.00	0.00	6.91
Solve Sub	2	40.68	60.84	312.76	30.52	70.83
Build Sub	2	13.22	17.72	33.78	2.37	13.40
Sum		55.97	79.82	349.08	292.59	95.23

Table 5: Arithmetic mean of computing times spent in each stage over the whole test set. The time spent in each stage and the time for initial setup of the algorithm, which includes computing the initial EV solution, are wall clock solution times.

	Level-ODA	Level	Benders-SC	Benders-MC	Benders-ODA
Time Stage 1	1.40	1.28	2.69	258.58	4.27
Time Stage 2	9.49	13.33	50.27	4.79	19.50
Time Setup	0.31	0.31	0.31	0.23	0.31
Sum	10.88	14.61	52.96	263.37	23.77

Measuring the wall clock computing time can only be done stage-wise. Table 5 shows these computing times. Comparing Benders-SC and Benders-MC, the times per task are reflected in the computing times per stage, where the second stage tasks are parallelized. Thus Benders-MC takes more than five times longer than Benders-SC to solve all problems. Regarding on-demand accuracy and level decomposition, an increase of 9% in first stage solution time is more than offset by a larger decrease of 29% in second stage solution time, which leads to a decrease of 25 % in overall solution time for the level method with on-demand accuracy compared to the level method.

The summed wall clock solution times for the individual threads in Table 4 also allow to measure the effect of parallelization on the relative performance of the algorithms. As only the building and solving

of second stage subproblems, as well as the computation of on-demand accuracy cuts is done in parallel, algorithms that spend more time in the second stage benefit more from parallelization than those algorithms that do not. This is particularly noteworthy for single-cut Benders and multi-cut Benders, by comparing the respective parallel wall clock computing time given in Table 5 with the summed wall clock computing times over all threads in Table 4. Note that the times in Table 4 can only be used as a rough estimator for the sequential computing time, as the solver used eight threads, but has only four physical cores available, thus some overhead is included in these results. With this caveat in mind, the times from Table 4 show that Level-ODA runs in 16% of Benders-SC, but the wall clock computing time gives only an improvement of 21%. Thus parallelization diminishes the advantage of level decomposition with and without on-demand accuracy, compared with Benders decomposition.

Comparing the deak test set

The on-demand accuracy approach proposed by [18] was initially evaluated only on the deak testset [5]. Therefore, we tested our algorithm also only on the deak test set, but with instances with 10, 20, 30, 40, 50, 80, 100, 150, 200, 250, 300, 350, 400, 450 and 500 scenarios instead of just the instances with 400, 450 and 500 scenarios, which we selected for our test set above. The performance profile for the deak test set is shown in Figure 2. Our results, given in Table 6, confirm the results of our more general test case above

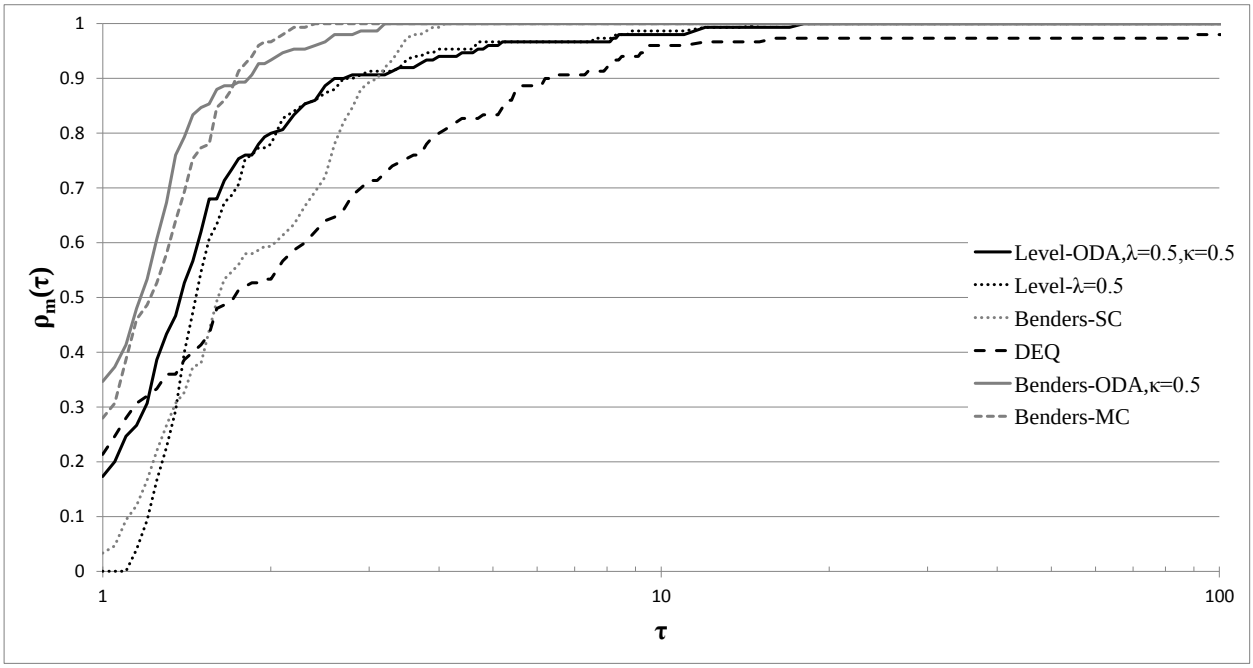

Figure 2: Performance profile of different algorithms on the deak test set.

and the results of Oliveira and Sagastizábal [18]. However, it is interesting that the multi-cut L-shaped method is faster than the level method with on-demand accuracy on this test set. This can be explained by the size of the test instances, which are rather small problems. Therefore cut proliferation does not become a problem. The multi-cut method is also faster on the smaller scenario instances, but slower on the larger scenario instances than the level method with on-demand accuracy. Due to many small scenario instances, the multi-cut method is the fastest method overall on this test set. Comparing only the single-cut algorithms, level decomposition with on-demand accuracy approach is faster than both level decomposition and single-cut Benders, but slower than Benders-ODA.

Table 6: Computational results for the full deak test set. Computing times in seconds for different algorithms on the test set. The computing times are both given as absolute values and as as a percentage of the computing time of single-cut Benders decomposition (Benders-SC), for the arithmetic mean (AM), geometric mean (GM) and the shifted geometric mean (SGM). The compared methods are level decomposition with on-demand accuracy (Level-ODA), level decomposition (Level), multi-cut Benders decomposition (Benders-MC), Benders decomposition with on-demand accuracy (Benders-ODA) and the deterministic equivalent (DEQ). The Cplex barrier method solved three problems incorrectly, 60x60-1_80,40x40-1_80 and 60x20-1_250. These problems were excluded for the computation of the results for the DEQ.

	AM	% Benders-SC	GM	% Benders-SC	SGM+10	% Benders-SC
Level-ODA	0.30	66	0.23	88	0.29	68
Level	0.33	75	0.25	95	0.33	76
Benders-SC	0.45	100	0.26	100	0.43	100
Benders-MC	0.27	61	0.19	70	0.27	62
Benders-ODA	0.24	54	0.18	69	0.24	55
DEQ	0.50	111	0.31	116	0.48	110

Table 7: Iteration counts for the full deak test set.

	AM	% Benders-SC	GM	% Benders-SC	SGM+10	% Benders-SC
Level-ODA	47.76	62	39.97	76	41.38	73
Level	43.69	57	36.51	69	37.89	67
Benders-SC	77.20	100	52.54	100	56.61	100
Benders-MC	9.16	12	8.75	17	8.97	16
Benders-ODA	84.74	110	57.68	110	61.85	109

The iteration counts presented in Table 7 and 8 and the detailed computing times in Table 9 and 10 explain these findings. The multi-cut method needs on average only 9 iterations, whereas the level method with on-demand accuracy needs 48. The lesser time spent on solving the master problem in the level method with on-demand accuracy is not small enough to offset the computing times incurred by solving more second stage subproblems than multi-cut Benders, due to the higher number of iterations. In comparison with level decomposition without on-demand accuracy, the reduction in substantial iteration is enough to offset the increase in overall iterations.

Benders-ODA is faster than Level-ODA for this test set, although it needs 77% more iterations. This is due to the fact that Benders-ODA does not have to solve a projection problem. In the sequential case, the relative order of Benders-ODA and Level-ODA would most likely change, according to the times in Table 9.

Table 8: Substantial iteration counts for the full deak test set.

	AM	% Benders-SC	GM	% Benders-SC	SGM+10	% Benders-SC
Level-ODA	22.01	29	20.34	39	20.85	37
Level	43.69	57	36.51	69	37.89	67
Benders-SC	77.20	100	52.54	100	56.61	100
Benders-MC	9.16	12	8.75	17	8.97	16
Benders-ODA	16.30	21	14.78	28	15.44	27

Table 9: Arithmetic mean of computing times spent at distinct tasks over the full deak test. The time for the tasks is wall clock time of all threads, i.e., summed over all threads.

Task	Stage	Level-ODA	Level	Benders-SC	Benders-MC	Benders-ODA
Solve Master	1	0.01	0.01	0.02	0.13	0.03
Solve Projection	1	0.09	0.08	0.00	0.00	0.00
Solve ODA	2	0.02	0.00	0.00	0.00	0.03
Solve Sub	2	0.45	0.81	1.87	0.30	0.53
Build Sub	2	0.03	0.05	0.13	0.01	0.02
Sum		0.60	0.95	2.02	0.45	0.62

Table 10: Arithmetic mean of computing times spent in each stage. The time spent in each stage and the time for the initial setup of the algorithm, which includes computing the initial EV solution, are wall clock solution times.

	Level-ODA	Level	Benders-SC	Benders-MC	Benders-ODA
Time Stage 1	0.11	0.10	0.03	0.14	0.04
Time Stage 2	0.11	0.16	0.34	0.06	0.13
Time Setup	0.07	0.07	0.07	0.07	0.07
Sum	0.23	0.26	0.38	0.20	0.17

5 Conclusion

In this paper, we show how the concept of on-demand accuracy can be incorporated into solution methods for two-stage stochastic programming problems based on Benders' decomposition principle. We devised a regularized and an unregularized variant of the single-cut L-shaped method using the on-demand accuracy approach to distinguish between substantial and unsubstantial iterations. An interation is called substantial if the current optimal objective function value of the master problem falls below a certain descent target. Only in substantial iterations a conventional optimality cut is constructed. In unsubstantial iterations, an approximated cut is added to the master problem that can be obtained without solving the second-stage subproblems.

The computational results clearly show that for the unregularized variant, as expected, the total number of iterations increases by 20% on average which can be explained by the decreased accuracy of the cuts in the master problem. For the regularized variant this increase is only roughly 5%. However, in both cases the slight increase in iterations in by far outweighed by a drastic decrease in average computing time per iteration as only 16% of the iterations are substantial. This resulted in a decrease of overall solution time of 55% on average compared to the conventional single-cut L-shaped method. Combining the on-demand accuracy approach with regularization even led to a reduction in solution time of 79%. On the smaller sized problems of the deak test set, the single-cut variant with on-demand accuracy outperformed the multi-cut method that achieved by far the smallest iteration count. On the complete test set containing many large scale problems the multi-cut method was shown not to be competitive due to a high amount of solution time required to solve the master problems. Finally, we confirmed the result that regularization pays off on average. However, we showed that the effect is reduced in parallelized implementations as regularized methods spend less time solving second-stage subproblems which constitutes the step that benefits most from parallelization.

Future work includes the extension of the above ideas to the solution of risk averse problems and of stochastic programming problems with incomplete recourse, with binary variables on the first stage and with more than two stages. Furthermore, the combination of the on-demand accuracy approach with the

idea of cut consodidation proposed in [20] should be explored.

We think that the present line of research gives further insights into the effect of regularization in specialized solution methods. Such insights will be useful in developing specialized methods for special problems, like those arising in energy applications and smart grid models.

Acknowledgement

Christian Wolf's work has been supported by a grant from the Deutsche Forschungsgemeinschaft (DFG) under Grant No. SU136/8-1. Csaba Fábián's work has been supported by the European Union and Hungary and co-financed by the European Social Fund through the project TÁMOP-4.2.2.C-11/1/KONV-2012-0004: National Research Center for the Development and Market Introduction of Advanced Information and Communication Technologies. These sources of support are gratefully acknowledged.

References

[1] Achterberg, T. (2007) Constraint Integer Programming *Ph.D. thesis*, Technische Universität Berlin.

[2] Ariyawansa, K. A. and Felt, A. J. (2004) On a new collection of stochastic linear programming test problems *INFORMS Journal on Computing* **16**, 291-299.

[3] Birge, J.R. and F.V. Louveaux (1988). A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research* **34**, 384-392.

[4] Birge, J.R. and F.V. Louveaux (1997). *Introduction to Stochastic Programming*. Springer-Verlag, New York.

[5] Deák, I. (2011). Testing successive regression approximations by large-scale two-stage problems *Annals of Operations Research* **186**, 83-99.

[6] Dolan, E. D. and Moré, J. J. (2002) Benchmarking optimization software with performance profiles *Mathematical Programming* ,**91**, 201-213.

[7] Fábián, C.I. (2000). Bundle-Type Methods for Inexact Data. *Central European Journal of Operations Research* **8** (special issue, T. Csendes and T. Rapcsák, eds.), 35-55.

[8] Fábián, C.I. (2012). Computational aspects of risk-averse optimization in two-stage stochastic models. *Stochastic Programming E-Print Series,* 3-2013.

[9] Fábián, C.I. and Z. Szőke (2007). Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science* **4**, 313-353.

[10] Gassmann, H.I. (1990). MSLiP: a computer code for the multistage stochastic linear programming problem. *Mathematical Programming* **47**, 407-423.

[11] Holmes, D. (1995) A (PO)rtable (S)tochastic programming (T)est (S)et (POSTS). `http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html`.

[12] Kall, P. and Mayer, J.(1998). On testing SLP codes with SLP-IOR. *New trends in Mathematical Programming: Homage to Steven Vajda* , 115-135.

[13] Kiwiel, K.C. (1995). Proximal level bundle methods for convex nondifferentiable optimization, saddle-point problems and variational inequalities. *Mathematical Programming* **69**, 89-109.

[14] Koberstein, A., Lucas, C.,Wolf, C. and König, D. (2011) Modeling and optimizing risk in the strategic gas-purchase planning problem of local distribution companies. *The Journal of Energy Markets* **4**, 47-68

[15] Lemaréchal, C., A. Nemirovskii, and Yu. Nesterov (1995). New Variants of Bundle Methods. *Mathematical Programming* **69**, 111-147.

[16] Linderoth, J., Shapiro, A. and Wright, S. 2006 The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research* **142**, 215-241.

[17] Nemirovski, A. (2005). *Lectures in modern convex optimization.* ISYE, Georgia Institute of Technology.

[18] Oliveira, W. and C. Sagastizábal (2012). Level bundle methods for oracles with on-demand accuracy. *Optimization Online,* March 2012.

[19] Trukhanov, S., Ntaimo, L. and Schaefer, A. (2010) Adaptive multicut aggregation for two-stage stochastic linear programs with recourse *European Journal of Operational Research* **206**, 395-406.

[20] Wolf, C. and A. Koberstein (2013). Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested L-shaped method. *European Journal of Operational Research,* **230**, 143-156.

[21] Zverovich, V. C.I. Fábián, E.F.D. Ellison, and G. Mitra (2012). A computational study of a solver system for processing two-stage stochastic LPs with enhanced Benders decomposition. *Mathematical Programming Computation* **4**, 211-238.

Appendix: Detailed tables

Table 11: Problem dimensions

		Stage 1		Stage 2		DEQ		
Instance	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
20x20-1_400	deak	20	10	30	20	12020	8010	72483
20x20-1_450	deak	20	10	30	20	13520	9010	81533
20x20-1_500	deak	20	10	30	20	15020	10010	90583
20x40-1_400	deak	20	10	60	40	24020	16010	184083
20x40-1_450	deak	20	10	60	40	27020	18010	207083
20x40-1_500	deak	20	10	60	40	30020	20010	230083
20x60-1_400	deak	20	10	90	60	36020	24010	344083
20x60-1_450	deak	20	10	90	60	40520	27010	387083
20x60-1_500	deak	20	10	90	60	45020	30010	430083
40x20-1_400	deak	40	20	30	20	12040	8020	122725
40x20-1_450	deak	40	20	30	20	13540	9020	138025
40x20-1_500	deak	40	20	30	20	15040	10020	153325
40x40-1_400	deak	40	20	60	40	24040	16020	288325
40x40-1_450	deak	40	20	60	40	27040	18020	324325
40x40-1_500	deak	40	20	60	40	30040	20020	360325
40x60-1_400	deak	40	20	90	60	36040	24020	400325
40x60-1_450	deak	40	20	90	60	40540	27020	450325
40x60-1_500	deak	40	20	90	60	45040	30020	500325
60x20-1_400	deak	60	30	30	20	12060	8030	173127
60x20-1_450	deak	60	30	30	20	13560	9030	194677
60x20-1_500	deak	60	30	30	20	15060	10030	216227
60x40-1_400	deak	60	30	60	40	24060	16030	386727
60x40-1_450	deak	60	30	60	40	27060	18030	434977
60x40-1_500	deak	60	30	60	40	30060	20030	483227
60x60-1_400	deak	60	30	90	60	36060	24030	648727
60x60-1_450	deak	60	30	90	60	40560	27030	729727
60x60-1_500	deak	60	30	90	60	45060	30030	810727
100x20-1_400	deak	100	50	30	20	12100	8050	121416
100x20-1_450	deak	100	50	30	20	13600	9050	136466
100x20-1_500	deak	100	50	30	20	15100	10050	151516
stormG2_8	posts	121	185	1259	528	10193	4409	27424
stormG2_27	posts	121	185	1259	528	34114	14441	90903
stormG2_125	posts	121	185	1259	528	157496	66185	418321
stormG2_1000	posts	121	185	1259	528	1259121	528185	3341696
rand0_2000	rand	100	50	50	25	100100	50050	754501
rand0_4000	rand	100	50	50	25	200100	100050	1508501
rand0_6000	rand	100	50	50	25	300100	150050	2262501
rand0_8000	rand	100	50	50	25	400100	200050	3016501
rand0_10000	rand	100	50	50	25	500100	250050	3770501
rand1_2000	rand	200	100	100	50	200200	100100	3006001
rand1_4000	rand	200	100	100	50	400200	200100	6010001
rand1_6000	rand	200	100	100	50	600200	300100	9014001
rand1_8000	rand	200	100	100	50	800200	400100	12018001
rand1_10000	rand	200	100	100	50	1000200	500100	15022001
rand2_2000	rand	300	150	150	75	300300	150150	6758501

Table 11: Problem dimensions (continued)

		Stage 1		Stage 2		DEQ		
Instance	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
rand2_4000	rand	300	150	150	75	600300	300150	13512501
rand2_6000	rand	300	150	150	75	900300	450150	20266501
rand2_8000	rand	300	150	150	75	1200300	600150	27020501
rand2_10000	rand	300	150	150	75	1500300	750150	33774501
20-1000	sampling	63	3	764	124	764063	124003	4488063
20-2000	sampling	63	3	764	124	1528063	248003	8976063
20-3000	sampling	63	3	764	124	2292063	372003	13464063
gbd	sampling	4	2	12	7	12000004	7000002	28000008
LandS	sampling	17	4	10	5	6464267	3232129	17453492
ssn-1000	sampling	89	1	706	175	706089	175001	2373089
ssn-2000	sampling	89	1	706	175	1412089	350001	4746089
ssn-3000	sampling	89	1	706	175	2118089	525001	7119089
storm-1000	sampling	121	185	1259	528	1259121	528185	3341696
storm-2000	sampling	121	185	1259	528	2518121	1056185	6682696
storm-3000	sampling	121	185	1259	528	3777121	1584185	10023696
saphir_50	saphir	53	32	3924	8678	196253	433932	1136753
saphir_100	saphir	53	32	3924	8678	392453	867832	2273403
saphir_500	saphir	53	32	3924	8678	1962053	4339032	11366603
saphir_1000	saphir	53	32	3924	8678	3924053	8678032	22733103
sslp_10_50_50	SIPLIB	10	1	510	60	25510	3001	50460
sslp_10_50_100	SIPLIB	10	1	510	60	51010	6001	100910
sslp_10_50_500	SIPLIB	10	1	510	60	255010	30001	504510
sslp_10_50_1000	SIPLIB	10	1	510	60	510010	60001	1009010
sslp_10_50_2000	SIPLIB	10	1	510	60	1020010	120001	2018010
sslp_15_45_5	SIPLIB	15	1	690	60	3465	301	6835
sslp_15_45_10	SIPLIB	15	1	690	60	6915	601	13655
sslp_15_45_15	SIPLIB	15	1	690	60	10365	901	20475
airl	slptestset	4	2	8	6	204	152	604
airl2	slptestset	4	2	8	6	204	152	604
assets-small	slptestset	13	5	13	5	1313	505	2621
assets-large	slptestset	13	5	13	5	487513	187505	975021
4node-2	slptestset	52	14	186	74	424	162	1191
4node-4	slptestset	52	14	186	74	796	310	2127
4node-8	slptestset	52	14	186	74	1540	606	3999
4node-16	slptestset	52	14	186	74	3028	1198	7743
4node-32	slptestset	52	14	186	74	6004	2382	15231
4node-64	slptestset	52	14	186	74	11956	4750	30207
4node-128	slptestset	52	14	186	74	23860	9486	60159
4node-256	slptestset	52	14	186	74	47668	18958	120063
4node-512	slptestset	52	14	186	74	95284	37902	239871
4node-1024	slptestset	52	14	186	74	190516	75790	479487
4node-2048	slptestset	52	14	186	74	380980	151566	958719
4node-4096	slptestset	52	14	186	74	761908	303118	1917183
4node-8192	slptestset	52	14	186	74	1523764	606222	3834111
4node-16384	slptestset	52	14	186	74	3047476	1212430	7667967
4node-32768	slptestset	52	14	186	74	6094900	2424846	15335679

Table 11: Problem dimensions (continued)

		Stage 1		Stage 2		DEQ		
Instance	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
chem	slptestset	39	38	41	46	121	130	289
LandS	slptestset	4	2	12	7	40	23	92
env-aggr	slptestset	49	48	49	48	294	288	852
env-first	slptestset	49	48	49	48	1613521	1580592	4741764
env-loose	slptestset	49	48	49	48	294	288	852
env-imp	slptestset	49	48	49	48	784	768	2292
env-1200	slptestset	49	48	49	48	58849	57648	172932
env-1875	slptestset	49	48	49	48	91924	90048	270132
env-3780	slptestset	49	48	49	48	185269	181488	544452
env-5292	slptestset	49	48	49	48	259357	254064	762180
env-lrge	slptestset	49	48	49	48	294	288	852
env-xlrge	slptestset	49	48	49	48	403417	395184	1185540
phone	slptestset	8	1	85	23	2785288	753665	9863176
stocfor2	slptestset	15	15	96	102	6159	6543	26907

Table 12: Computing times for solution runs with level decomposition with on-demand accuracy (Level-ODA), level decomposition (Level), single-cut Benders (Benders-SC), multi-cut Benders (MC), Benders with on-demand accuracy (Benders-ODA) and the deterministic equivalent (DEQ). All times are given in seconds. Instances, where a method gave incorrect results are marked with †.

Instance	Level-ODA	Level	Benders-SC	Benders-MC	Benders-ODA	DEQ
20x20-1_400	0.2	0.1	0.1	0.1	0.1	0.4
20x20-1_450	0.1	0.1	0.1	0.1	0.1	1.0
20x20-1_500	0.1	0.1	0.1	0.1	0.1	0.3
20x40-1_400	0.2	0.2	0.1	0.2	0.1	0.6
20x40-1_450	0.2	0.1	0.1	0.2	0.1	0.7
20x40-1_500	0.2	0.2	0.2	0.2	0.1	0.6
20x60-1_400	0.4	0.5	1.0	0.5	0.5	0.5
20x60-1_450	0.4	0.5	1.0	0.6	0.5	0.6
20x60-1_500	0.5	0.6	1.2	0.6	0.6	0.8
40x20-1_400	0.2	0.2	0.2	0.3	0.1	0.4
40x20-1_450	0.2	0.2	0.2	0.3	0.1	0.4
40x20-1_500	0.2	0.2	0.2	0.4	0.2	0.4
40x40-1_400	0.3	0.2	0.2	0.2	0.2	1.4
40x40-1_450	0.3	0.2	0.2	0.3	0.2	3.0
40x40-1_500	0.2	0.3	0.3	0.3	0.2	1.5
40x60-1_400	0.5	0.8	1.4	0.7	0.7	1.0
40x60-1_450	0.5	0.8	1.5	0.8	0.7	1.1
40x60-1_500	0.6	0.8	1.9	1.0	0.8	1.0
60x20-1_400	0.5	0.4	0.5	0.5	0.4	0.4
60x20-1_450	0.4	0.4	0.6	0.6	0.4	0.6
60x20-1_500	0.4	0.5	0.6	0.6	0.4	0.6
60x40-1_400	0.5	0.8	1.4	1.0	0.6	0.9
60x40-1_450	0.6	0.8	1.7	1.0	0.7	1.0

Table 12: Computing times (continued)

Instance	Level-ODA	Level	Benders-SC	Benders-MC	Benders-ODA	DEQ
60x40-1_500	0.6	0.8	1.7	1.2	0.8	1.0
60x60-1_400	0.7	1.0	2.1	0.8	0.7	1.4
60x60-1_450	0.6	1.1	2.4	0.9	0.7	1.5
60x60-1_500	0.6	1.0	2.6	1.1	0.8	1.6
100x20-1_400	0.9	0.8	0.9	1.0	0.6	0.6
100x20-1_450	1.0	1.1	0.9	1.1	0.5	0.6
100x20-1_500	1.1	1.0	0.8	1.3	0.7	0.7
stormG2_8	0.1	0.1	0.1	0.1	0.1	0.1
stormG2_27	0.2	0.2	0.2	0.1	0.1	0.7
stormG2_125	0.3	0.4	0.4	0.2	0.3	2.4
stormG2_1000	1.2	1.8	2.6	1.4	1.2	22.7
rand0_2000	0.9	1.1	1.9	10.3	1.0	4.0
rand0_4000	1.4	1.6	3.2	26.5	1.8	11.0
rand0_6000	2.5	3.6	7.3	95.8	3.5	16.3
rand0_8000	2.7	3.6	9.8	148.5	4.2	31.1
rand0_10000	4.5	7.3	22.9	307.9	8.6	45.5
rand1_2000	4.5	6.2	27.1	60.5	12.6	14.1
rand1_4000	6.5	9.8	60.0	178.5	25.5	33.4
rand1_6000	8.1	13.4	72.2	320.6	29.0	54.5
rand1_8000	11.6	17.8	112.8	693.9	41.8	86.0
rand1_10000	15.1	26.2	149.9	1151.4	60.5	117.2
rand2_2000	10.3	18.6	168.0	124.3	73.6	41.4
rand2_4000	13.0	20.7	137.8	278.9	40.8	88.9
rand2_6000	22.0	34.6	280.8	768.3	81.0	152.7
rand2_8000	23.6	39.0	345.0	1082.3	85.1	213.7
rand2_10000	35.9	61.4	608.0	2207.9	184.0	269.1
20-1000	11.8	14.8	116.3	58.8	71.6	6.7
20-2000	20.2	40.8	204.6	154.7	217.0	13.2
20-3000	30.5	44.9	291.0	381.2	314.2	23.8
gbd	109.9	132.9	120.6	3600.0	74.9	245.5
LandS	74.0	180.7	229.1	3600.0	83.7	131.6
ssn-1000	4.9	10.0	244.1	4.2	87.8	20.4
ssn-2000	16.2	25.2	437.1	10.6	107.3	58.0
ssn-3000	26.8	40.1	620.8	17.6	129.8	†
storm-1000	1.4	2.0	3.2	1.5	1.3	22.9
storm-2000	2.6	3.9	5.9	3.5	2.6	52.4
storm-3000	3.6	4.8	8.7	7.3	3.5	83.4
saphir_50	29.4	38.6	57.3	12.6	31.4	7.5
saphir_100	45.7	54.0	80.1	16.5	50.3	†
saphir_500	134.5	166.1	257.1	60.8	141.2	799.3
saphir_1000	239.3	272.7	393.6	192.7	249.5	†
sslp_10_50_50	18.5	19.4	15.8	135.7	26.3	5.2
sslp_10_50_100	17.6	16.1	15.5	266.3	22.9	11.8
sslp_10_50_500	27.1	26.1	27.0	3600.0	30.6	215.0
sslp_10_50_1000	37.8	38.2	48.0	3600.0	41.6	754.4
sslp_10_50_2000	57.7	54.9	80.5	3600.0	65.2	3476.8
sslp_15_45_5	4.8	2.4	1.9	2.4	3.2	0.1

Table 12: Computing times (continued)

Instance	Level-ODA	Level	Benders-SC	Benders-MC	Benders-ODA	DEQ
sslp_15_45_10	7.3	6.3	5.4	4.7	11.6	0.5
sslp_15_45_15	7.3	7.1	5.0	5.1	11.7	0.3
airl	0.1	0.1	0.1	0.1	0.1	0.1
airl2	0.1	0.1	0.1	0.1	0.1	0.1
assets-small	0.1	0.1	0.1	0.1	0.1	0.1
assets-large	3.4	3.1	1.1	3.2	1.1	3.5
4node-2	0.1	0.1	0.1	0.1	0.1	0.1
4node-4	0.2	0.1	0.1	0.1	0.1	0.1
4node-8	0.1	0.2	0.1	0.1	0.1	0.1
4node-16	0.2	0.1	0.1	0.1	0.1	0.1
4node-32	0.2	0.2	0.2	0.1	0.1	0.2
4node-64	0.2	0.2	0.2	0.1	0.2	0.2
4node-128	0.3	0.3	0.3	0.2	0.2	0.1
4node-256	0.4	0.3	0.7	0.2	0.3	0.3
4node-512	0.5	0.6	1.3	0.4	0.6	1.5
4node-1024	0.6	1.0	2.2	0.5	0.9	3.0
4node-2048	1.4	1.8	6.1	1.9	2.1	2.8
4node-4096	2.8	3.5	16.0	5.7	4.8	6.9
4node-8192	5.0	6.2	33.7	28.4	10.0	11.2
4node-16384	12.2	15.2	69.3	103.3	18.9	22.1
4node-32768	21.8	27.3	145.3	458.1	29.8	52.7
chem	0.1	0.1	0.1	0.1	0.1	0.1
LandS	0.1	0.1	0.1	0.1	0.1	0.1
env-aggr	0.1	0.1	0.1	0.1	0.1	0.1
env-first	0.1	0.1	0.1	0.1	0.1	0.5
env-loose	0.1	0.1	0.1	0.1	0.1	0.1
env-imp	0.1	0.1	0.1	0.1	0.1	0.1
env-1200	0.5	0.5	0.2	0.7	0.2	2.4
env-1875	0.7	0.7	0.4	2.7	0.4	2.2
env-3780	1.4	1.4	0.8	7.3	0.7	5.8
env-5292	1.8	1.9	1.0	18.1	1.0	9.1
env-lrge	2.8	2.8	1.5	17.2	1.7	20.1
env-xlrge	11.0	11.2	5.9	215.3	6.1	†
phone	1.1	1.1	1.4	1.8	1.3	18.2
stocfor2	0.1	0.1	0.1	0.1	0.1	0.1